AFRL-IF-WP-TR-2005-1555

MODEL-BASED INTEGRATED SIMULATION (MILAN)

Viktor K. Prasanna
Cauligi S. Raghavendra
Akos Ledeczi

University of Southern California
Los Angeles, CA 90089-2562

MAR 2005

Final Report for 29 June 2000 – 31 December 2004

INFORMATION DIRECTORATE
AIR FORCE RESEARCH LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7334

NOTICE

PAO Case Number: AFRL/WS-05-930, 18 Apr 05

THIS TECHNICAL REPORT IS APPROVED FOR PUBLICATION.

[illegible]

JONG S. HWANG, Program Monitor

[illegible]

ROBERT A. EHRET, Chief
Collaborative Simulation Technology & Applications Branch
Information Systems Division
Information Directorate

REPORT DOCUMENTATION PAGE		*Form Approved OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* March 2005	2. REPORT TYPE Final	3. DATES COVERED *(From - To)* 29 Jun 2000 – 31 Dec 2004
4. TITLE AND SUBTITLE Model-based Integrated Simulation (MILAN)		5a. CONTRACT NUMBER F33615-00-C-1633
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER 69199F
6. AUTHOR(S) Viktor K. Prasanna Cauligi S. Raghavendra Akos Ledeczi		5d. PROJECT NUMBER ARPI
		5e.TASK NUMBER FS
		5f. WORK UNIT NUMBER A4
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Southern California Los Angeles, CA 90089-2562		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) INFORMATION DIRECTORATE AIR FORCE RESEARCH LABORATORY AIR FORCE MATERIEL COMMAND WRIGHT-PATTERSON AFB, OH 45433-7334		10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/IFSD
		11. SPONSOR/MONITOR'S REPORT NUMBER(S) AFRL-IF-WP-TR-2005-1555
12. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		
13. SUPPLEMENTARY NOTES This document contains color.		

14. ABSTRACT

The motivation for the Model-based Integrated Simulation (MILAN) project is to develop an extensible modeling, simulation, and design space exploration framework for the design of latency and energy efficient embedded systems for signal processing applications. Design of embedded systems requires minimization of energy dissipation (to maximize battery life) while meeting a given latency constraint (typically real-time constraints). While until now Application Specific Integrated Circuits (ASICs) were considered the primary choice for low power high performance embedded systems, the recent advances in the design of general purpose processors (GPP), digital signal processors (DSP), field programmable gate arrays (FPGAs), and memories have provided viable commercial-off-the-self (COTS) alternatives to ASICs. These devices are designed using low-leakage process and support a number of low power operating and standby states, dynamic voltage and frequency scaling, among others to support energy optimization. In the MILAN project, we focus on signal processing applications that process a stream of input frames while meeting a given latency constraint for the processing of a single frame. MILAN is a joint effort by the University of Southern California and Vanderbilt University and is supported by the DARPA Power Aware Computing and Communication Program.

15. SUBJECT TERMS:
Embedded systems, simulation, design space exploration, reconfigurable systems

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Jong S. Hwang
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER *(include area code)*
Unclassified	Unclassified	Unclassified	SAR	34	937- 904-9048

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. 239.18

TABLE OF CONTENTS

1 Introduction

The motivation for the Model-based Integrated Simulation (MILAN) project is to develop an extensible modeling, simulation, and design space exploration framework for the design of latency and energy efficient embedded systems for signal processing applications. Design of embedded systems requires minimization of energy dissipation (to maximize battery life) while meeting a given latency constraint (typically real-time constraints)[14]. While until now Application Specific Integrated Circuits (ASICs) were considered the primary choice for low power high performance embedded systems, the recent advances in the design of general purpose processors (GPP), digital signal processors (DSP), field programmable gate arrays (FPGAs), and memories have provided viable commercial-off-the-self (COTS) alternatives to ASICs [26]. These devices are designed using low-leakage process and support a number of low power operating and standby states, dynamic voltage and frequency scaling, among others to support energy optimization. Therefore, COTS devices such as Intel PXA 255 [11], IBM PowerPC 405 LP [9], Actel ProASIC [1], TI C5000 series DSPs [41], and Micron Mobile SDRAM [21] are being considered as candidate components (processing and memory) for low power high performance embedded design.

However, in terms of performance and flexibility, each class of components has its own advantages and disadvantages. For example, an ISA-based embedded processor (GPP, DSP, or micro-controller) is software programmable, possibly low power, but may not meet the high performance needs of some signal processing application [34]. In contrast, FPGAs support high degree of parallelism resulting in higher performance but are not energy efficient and may not be suitable for control intensive applications. Additionally, applications also enforce specific functional requirements, which require specific hardware capabilities. For example, some signal processing applications exist which requires high precision floating-point operations [38]. Such applications require the use of floating-point processors, which may not be energy efficient, or software emulation on fixed-point processors, which may not be latency efficient. Therefore, during the design of low power embedded systems a number of commercial-off-the-self devices must be evaluated to identify the suitable hardware for a given signal processing application. With the availability of multiple implementation platforms such as FPGAs, traditional processors, and DSPs, a designer not only needs to identify suitable platforms but also appropriate hardware/software partitioning and mapping onto those platforms. In addition, other capabilities that play a significant role, especially for energy efficient design, are reconfiguration, dynamic voltage scaling, and choice of low power operating states. While minimizing energy dissipation, our focus is on maximizing battery life. Therefore, energy optimization with respect to the behavior of the embedded system while processing a single input is not adequate. Energy models based on the processing of a single input do not include the behavior of the embedded system when it is idle. Due to quiescent power, energy dissipation when a system is idle can be significant. Therefore, low power embedded system design requires design space exploration based on duty cycle specification to identify suitable device activation schedule that meets the given latency requirements while minimizing energy dissipation when the devices are active and when they are not [26]. Duty cycle is the proportion of time during which a system is operated. Such specification allows modeling of a period of execution as alternate active and inactive phases. Energy dissipation (e.g. due to leakage current), especially for systems with low duty cycle, during the inactive phases can contribute significantly to the overall energy dissipation of the system. Therefore, the tradeoff between the performance cost of shutting down and starting up a device and the performance cost of remaining idle needs to be considered during system design.

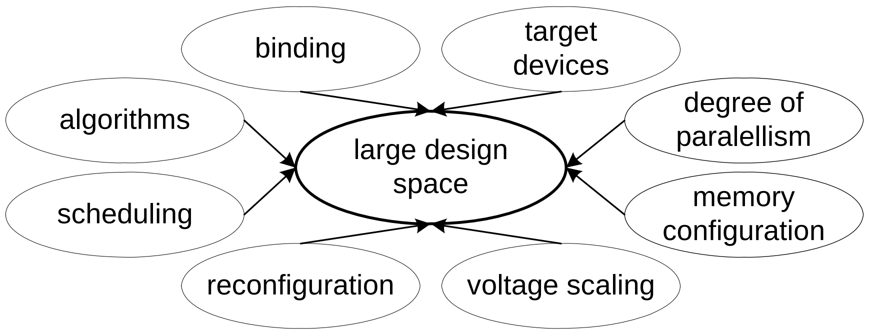

Figure 1: Large design space

In the MILAN project, we focus on signal processing applications that process a stream of input frames while meeting a given latency constraint for the processing of a single frame [33]. Examples of such applications include mobile base stations for software-defined radio, target detection and tracking systems, and space applications. We assume that the application logic and therefore the performance are independent of the data contained in the input frames. Such signal processing applications can be modeled as data flow graphs [18]. A data flow graph is a directed acyclic graph where the nodes in the graph represent the tasks (or kernels) and the directed edges connecting the nodes represent the dependency (order of execution and data flow) among the tasks [19]. The latency (and energy) constraints are specified based on the end-to-end performance of a sub-graph of the complete data flow graph where the sub-graph has one source node and one sink node. Additional advantage of a data flow graph is that such representation of an application allows it to be statically scheduled using topological sort and greedy scheduling.

Thus, many choices/tradeoffs are available during energy and latency efficient system design. However, a large number of choices during application design results in a large design space (Figure 1) that must be traversed efficiently to identify the designs that meet the performance requirements. A number of tools and simulators exist for the design of embedded system. SimpleScalar[36], PowerAnalyzer [31], ARMulator [4], Mambo [6], SimplePower [37], JouleTrack [39], Xpower [44], DESERT [27] are some such tools. However, due to lack of a standard interface, integration of these tools is extremely difficult. In addition, execution speed can vary widely among these tools, which makes it even harder to integrate (e.g. ~ 200 kilo instructions per second for SimpleScalar and < 10 instructions per second for FPGA simulators). Therefore, the focus of the MILAN project is to develop a design framework that allows integration of a variety of widely used simulators for candidate embedded devices through a unified interface and integrate a number of simulators into the framework. In addition, the framework would be extensible such that additional simulators can be easily integrated.

In order to facilitate specification of the application, target hardware, design and performance constraints, and integration of design tools, in the MILAN project, we have developed a set of metamodels (modeling paradigms). These metamodels define a domain-specific modeling language that is used to capture system models [17][18][23][27][28]. Our models capture application specifications, details of the candidate hardware for embedded systems, performance and design constraints, and deployment scenarios wrt. duty cycle specification. Generic Model provides an abstraction of the embedded systems that identifies the key architectural features that can be exploited for performance and energy optimization [22]. These features include, operating states, average power dissipation in each state, and state transition costs in terms of latency and energy dissipation. Application model is developed by enhancing the data flow graphs to capture choice of implementations for each task, and state transitions (e.g. reconfiguration or dynamic voltage scaling) between task executions [18]. MILAN supports both synchronous and asynchronous data flow graphs. Such representation allows us to define application design problem as a combinatorial algorithm that can be solved in an efficient manner using e.g. dynamic programming or design space exploration tools such as DESERT and HiPerE. A mapping model is also developed to specify the mapping between the application and resource models. MILAN also includes a set of models to specify hardware designs that can be used to specify FPGA based designs as well as generic hardware designs (not necessarily implementable using FPGAs) specified using HDL (hardware definition language) such as VHDL. Our modeling technique also allows us to develop library of models promoting reuse.

To address the problem of efficient exploration of a large design space, the MILAN project has developed a hierarchical design space exploration methodology [26]. Our methodology consists of two phases. The first phase uses a pruning technique that evaluates the initial design space and prunes it to a smaller set of designs based on the performance requirements. The pruning technique operates on the high-level models that specify the target application, candidate processing devices and memories, and performance constraints. The second phase uses a high-level estimation tool and low-level simulators to perform hierarchical simulation. Hierarchical simulation uses low-level simulators to perform component specific simulations for a given design. The component specific estimates are combined using the high-level estimator to generate system-wide performance estimates for a complete heterogeneous embedded system based on a duty cycle specification. In our methodology, hierarchical simulation is used to evaluate the designs identified in the first phase. The high-level estimation tool used for hierarchical simulation operates at a higher level of abstraction than a typical low-level simulator such as cycle-accurate, register-transfer level, or even instruction-set simulators. For example, the high-level estimation tool discussed in this thesis requires application input as a data flow graph, which is at a comparatively higher level of abstraction than say ``C'' as required by SimpleScalar [36]. Through the integration of simulators, estimator, and pruning techniques, our methodology exploits the speed versus accuracy tradeoffs to perform faster, compared to simulation only, and more accurate, compared to optimization techniques, evaluation of large design spaces.

MILAN is a joint effort by the University of Southern California and Vanderbilt University and is supported by the DARPA Power Aware Computing and Communication Program through contract number F33615-C-00-1633 monitored by Wright Patterson Air Force Base.

1.1 MILAN overview

MILAN is implemented using Model Integrated Computing (please refer to [9][14][40] for more information). MIC employs domain-specific models to represent the system being designed. These models are then used to automatically synthesize other artifacts. This approach speeds up the design cycle, facilitates the evolution of the application, and helps system maintenance, dramatically reducing costs during the entire lifecycle of the system. MIC is implemented by the Generic Modeling Environment (GME), a metaprogrammable toolkit for creating domain-specific modeling environments [9]. GME employs metamodels that specify the modeling paradigm of the application domain. The modeling paradigm contains all the syntactic, semantic, and presentation information regarding the domain – which concepts will be used to construct models, what relationships may exist among those concepts, how the concepts may be organized and viewed by the modeler, and rules governing the construction of models. The modeling paradigm defines the family of models that can be created using the resultant modeling environment. The metamodels specifying the modeling paradigm are used to automatically configure GME for the domain.

For the MILAN project, GME is used primarily for model-building. The models take the form of graphical, multi-aspect, attributed entity-relationship diagrams. The static semantics of a model are specified by OCL constraints [9] that are part of the metamodels. They are enforced by a built-in constraint manager during model building time. The dynamic semantics are applied by the model interpreters, i.e. by the process of translating the models to source code, configuration files, database schema or any other artifact the given application domain calls for.

The MILAN architecture is depicted in Figure 2. The design-space of a system is captured by multiple-aspect, hierarchical, primarily graphical models in GME. The three main categories of models specify the desired application functionality, available hardware resources and non-functional requirements in the form of explicit constraints. These complex models typically specify an exponentially large design-space. However, only a subset of this space satisfies all the constraints. A symbolic constraint satisfaction methodology is applied to explore and prune the design-space. Once a single design has been selected, model interpreters translate the models into the input of the selected simulators. Simulation results need to be incorporated back in the models. For some simulators this will necessarily be a human-in-the-loop process, while for others the procedure can be automated.

The final component in the MILAN architecture is System Synthesis. Notice that this step is similar to driving simulators. Instead of targeting the execution model of a simulation engine, the synthesis process needs to generate code that complies with the runtime semantics of a runtime system. Just like there is a need to support multiple simulators, MILAN needs to support multiple target runtime systems. Currently, MILAN is more focused on providing a simulation integration environment than providing system synthesis capabilities.

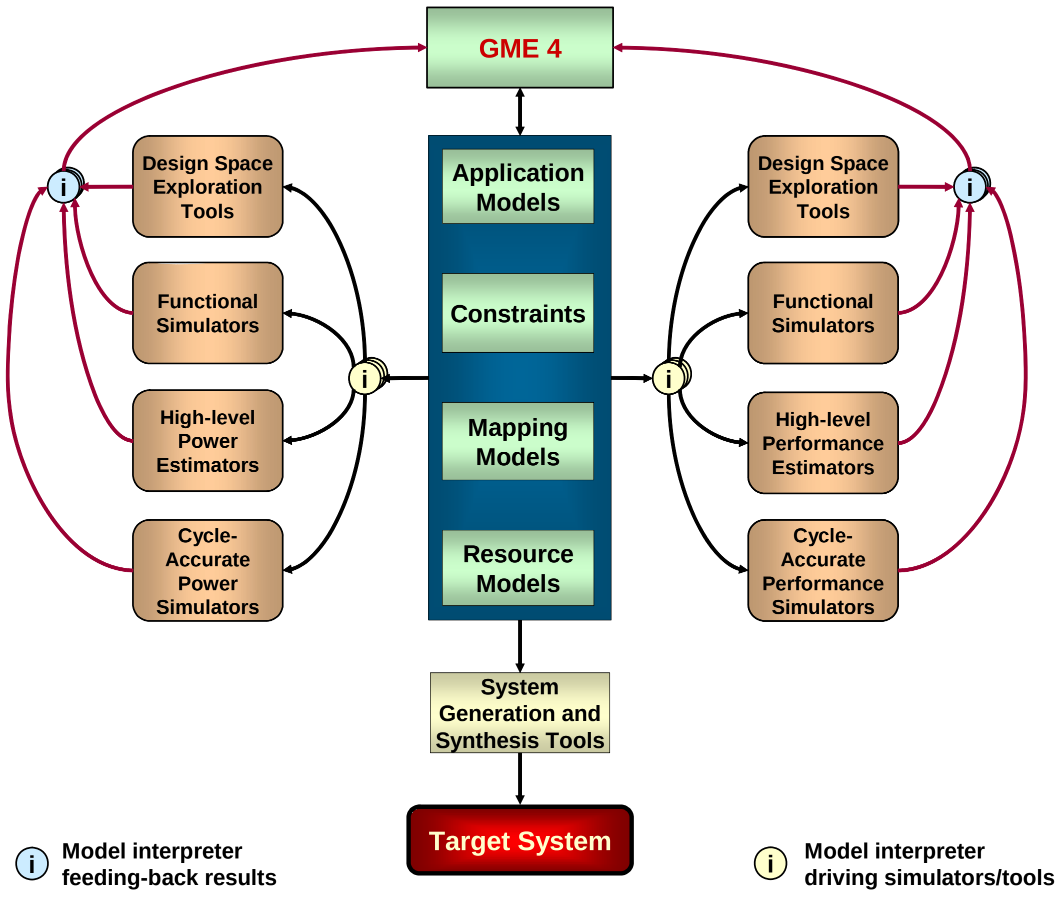
GME 4
Application Models
Constraints
Mapping Models
Resource Models
Design Space Exploration Tools
Functional Simulators
High-level Power Estimators
Cycle-Accurate Power Simulators
Design Space Exploration Tools
Functional Simulators
High-level Performance Estimators
Cycle-Accurate Performance Simulators
System Generation and Synthesis Tools
Target System
i
Model interpreter feeding-back results
i
Model interpreter driving simulators/tools

Figure 2: MILAN Architecture

2 Summary of Contributions

2.1 *Hierarchical Design Space Exploration*

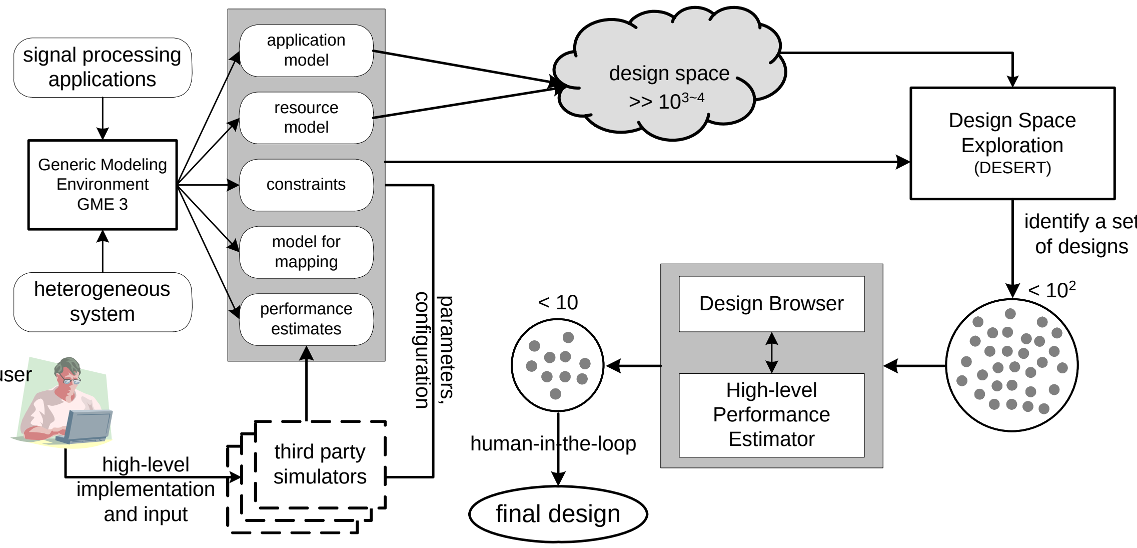
signal processing applications
Generic Modeling Environment GME 3
heterogeneous system
user
high-level implementation and input
application model
resource model
constraints
model for mapping
performance estimates
third party simulators
parameters, configuration
design space >> 10^3~4
Design Space Exploration (DESERT)
identify a set of designs
< 10^2
Design Browser
High-level Performance Estimator
< 10
human-in-the-loop
final design

Figure 3: MILAN Design Flow

MILAN is a model based integrated simulation environment for embedded system design and optimization through integration of various simulators and tools into a unified environment. Using the MILAN environment, the designer formally models the target application, underlying hardware, and constraints (latency, throughput, energy dissipation, etc.) through a graphical interface provided by MILAN. The models are stored in a model database. The model information is translated through model interpreters into suitable input formats required by the integrated simulators. MILAN adopts Model Integrated Computing (MIC) as the core design technology. The Generic Modeling Environment (GME) is a configurable graphical tool suite supporting MIC. GME allows the designer to create domain-specific models. A metamodel (modeling paradigm) is a formal description of model construction semantics. Once the metamodel is specified by the user, it can be used to configure GME itself to present a modeling environment specific to the problem domain. MILAN defines its own metamodel that is used to configure GME to provide the MILAN design environment. Every target system is specified in MILAN as a model. Model interpreters are the software components that translate the information captured in the models based on the input format required by the integrated tools and simulators.

2.1.1 MILAN Design Flow for Hierarchical Design Space Exploration

In this section, we provide a brief overview of the MILAN design flow. The user Manual (see Appendix) and the tutorials (included in the MILAN software releases) provide additional details about the MILAN design flow.

MILAN design flow consists of modeling, performance estimation, and design pace exploration. The user initiates the design process by modeling the application and the target architecture. Application modeling involves application specification as a data-flow graph with alternatives. The alternatives refer to various implementation choices available for an application task. The functional specification of the target system specifies the structure of the data-flow graph and the choice of implementations specifies the alternatives. For multi-rate applications, while modeling an application in MILAN, the designer can specify the rate of execution for each application task. A rate of r for a task refers to one execution per r input frames. MILAN supports hierarchical modeling that enables hierarchical specification of the data flow graph making it easier to manage and analyze an application model. Functional simulation and verification may be done iteratively with application modeling. To enable functional simulation, MILAN supports generation of high-level source code in C and Matlab and integration of functional simulators. MILAN also supports specification of input stimulus at the source tasks and output processing logic at the sink tasks. These capabilities are also exploited for simulation using integrated simulators to estimate performance. MILAN supports both synchronous and asynchronous data flow graph based application modeling. For asynchronous modeling MILAN provides a run-time kernel for the execution of an asynchronous implementation.

Resource modeling involves modeling of the target architecture. The modeling paradigm is based on the GenM and the augmented FSM model [27]. The user identifies key components and features of the target architecture that can be exploited for optimization and models them in MILAN. In addition, the user also models various states and state transition costs associated with different components such as reconfigurable devices, processors supporting DVS, and power aware memories using the augmented finite state machine model.

Finally, the user describes possible mappings for each task alternatives and different performance or compositional constraints that the system needs to satisfy. A mapping is a relation between an application task and a processing component. Performance constraints are based on the latency and energy dissipation requirements given as input. Design constraints capture requirements of mapping and composition of components in case of device selection. Constraints on composition restrict the composition of alternate processing components. For example, given a set of choices that includes two traditional processors, one such constraint can be “a valid system may contain one of the processors but not both”. Before we can perform design space exploration, we need to populate the design space described above using the performance estimates for all the mappings specified in the model. MILAN is a simulator integration environment. Hence, if appropriate simulators are integrated, MILAN has the capability to perform automatic simulation (using specified implementation and sample input) and update the model for mapping using the simulation results.

Once the complete system is modeled, the user invokes the design space exploration (DSE) tools. A DSE tool rapidly identifies a set of design that satisfies all the constraints. Currently, MILAN provides Design Space Exploration Tool, DESERT, as the primary DSE tool. DESERT is a constraint-based rapid design space exploration tool, developed at ISIS, Vanderbilt University [29]. Our experience with DESERT shows that we can prune a design space with approximately 10^{20}~10^{40} designs in order of minutes. DESERT uses symbolic methods based on Ordered Binary Decision Diagrams (OBDDs) for constraint satisfaction. We have also integrated other techniques based on dynamic programming to provide additional DSE options to the user. One such technique targets applications that can be modeled as a linear array of tasks. Given a linear array of tasks, execution cost of each task on a target reconfigurable device, and reconfiguration cost, the technique can identify a set of mapping with minimum latency or energy.

Design space exploration techniques are optimization heuristics based on a high-level model of the target system. Therefore, the accuracy of the result depends on the assumptions made during high-level modeling. Hence, we use hierarchical simulation to further evaluate the designs identified by the DSE techniques (Figure 3). Hierarchical simulation is a design evaluation technique based on simulation of the designs. Hierarchical simulation implies simulation of a task or a set of tasks at different levels of abstraction by exploiting the availability of simulators at different levels of abstraction. Multiple abstraction levels make it possible to control the speed and accuracy of the simulation results. Such flexibility enables a stepwise refinement approach in which initially fast simulators are used to efficiently explore a large number of designs and later more accurate simulators are used to facilitate a detailed evaluation of the designs. MILAN facilitates hierarchical simulation through seamless integration of different simulators and the ability to invoke all integrated simulator from a single modeling environment. However, due to the complexity of the target heterogeneous systems, either there is a lack of appropriate system-level simulator that can simulate the complete system, or if such a simulator exists it is prohibitively expensive in terms of simulation time. High-level Performance Estimator (HiPerE) tool, currently integrated into MILAN, addresses both the above issues.

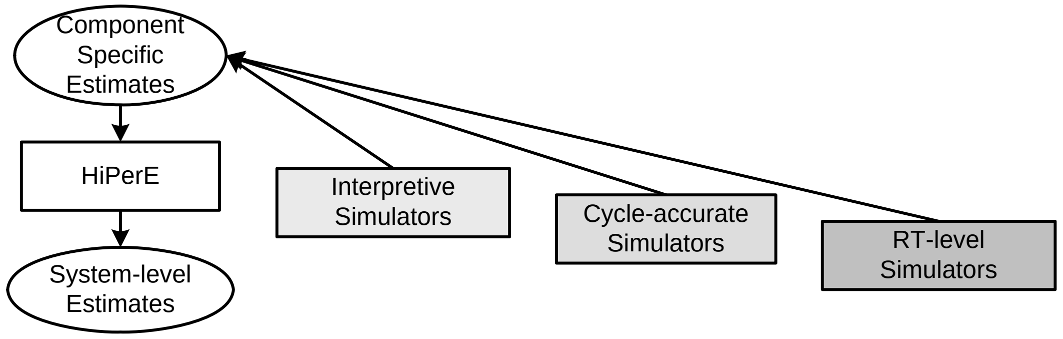

Figure 4: Hierarchical Simulation

Given a design, HiPerE evaluates system-level energy dissipation and latency. In order to provide a rapid estimate, HiPerE operates at the task level abstraction of the application. In addition to the task execution cost, various other aspects considered by HiPerE for accurate performance estimation are data access cost, parallelism in the system, energy dissipation when a component is idle, and state transition cost. Our results for signal processing applications show that HiPerE estimates are within 8% of the estimates using low-level simulations. HiPerE also produces an activity report that provides a coarse, task-level trace of execution time behavior of the application. An activity report contains processor specific task schedules, state transitions, and idle time or slack in between executions. The user can exploit the activity report to identify bottlenecks and further optimization opportunities. Duty-cycle in the context of application execution refers to the proportion of time during which a component, device, or system is operated. Support for duty-cycle includes estimation of performance for a length of time or number of execution instances while taking into account, start up and shut down cost, idle energy dissipation, and rate of input. HiPerE supports performance estimation of a design for a given duty-cycle. Additional details about HiPerE can be found in the MILAN User Manual 1.1 (see Appendix)

2.1.2 Key Ideas of Our Methodology

In order to discuss our methodology, we define problem domain to refer to a class of system design problems. Given an application that can be modeled using a DFG and a set of target devices, problem domain refers to all system design problems that can be defined based on the application and the target devices. For example, a linear array of tasks and a processor that supports dynamic voltage scaling define a problem domain which includes the following system design problem, "identify appropriate voltage setting for each task such that overall energy dissipation is minimized". Another problem may require minimization of energy while a given latency constraint is satisfied. A set of parameters are associated with each problem domain. A parameter can be a variable or a constant. For example, for the problem domain discussed above, some of the parameters are task execution cost for each voltage setting, operating voltage, voltage scaling cost, cost of data access due to the use of a memory, choice of memory devices, etc. Among the parameters, operating voltage is a variable parameter and task execution cost per operating voltage is constant. For a given problem domain, there exist several optimization heuristics, high-level estimators, and simulators that can be used to perform design space exploration. We define *parameter coverage* as a metric to compare different design space exploration (DSE) techniques applicable to a problem domain. For a given solution, be it an optimization heuristic or an estimation/simulation tool, parameter coverage refers to the set of parameters that are considered by each solution while estimating performance or performing design space exploration. Higher parameter coverage refers to a larger set of parameters and results in higher accuracy but can potentially be time consuming during DSE. A low-level simulator is an example of a performance estimation tool with high parameter coverage (e.g. SimpleScalar). In contrast, optimization heuristics, due to high-level of abstraction, tend to have lower parameter coverage.

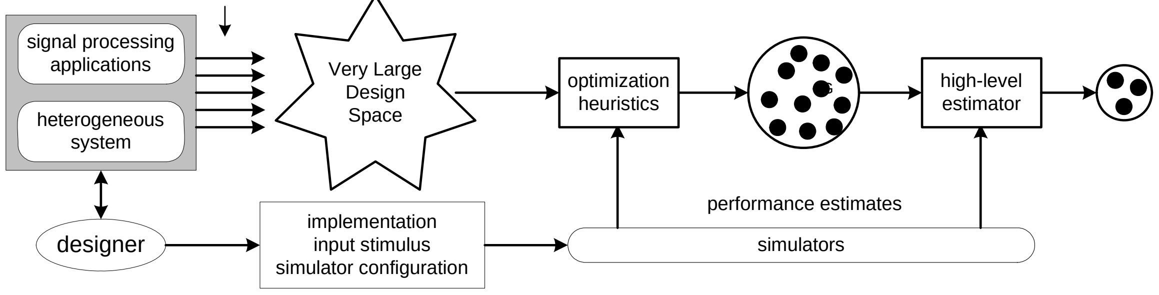

Figure 5: Hierarchical Design Space Exploration

Given a problem domain, hierarchical design space exploration is defined as a two step process (Figure 4). The first step uses an pruning heuristic that generates a set of designs meeting the given constraints. The second step consists of a suitable high-level performance estimator that evaluates the performance of any design that is a potential solution for the given problem domain. The high-level estimator we discuss in this report, HiPerE, uses an interpretive simulation based approach to estimate performance. Therefore, the high-level performance estimator has higher parameter coverage than the pruning heuristic. Hence, HiPerE can cover the parameters not included in the model used by the pruning heuristic. As a result, our methodology can explore a larger design space than an optimization heuristic only DSE scheme. Hierarchical design space exploration assumes the availability of appropriate simulators to estimate the model parameters required by the pruning heuristics and the high-level estimators. Examples of such parameters are the performance cost of various mappings or operating state transition costs etc. Some of the parameters can also be obtained from the data sheets provided by the device vendors. We also assume that appropriate input such as implementations in scripts or languages supported by the simulators, input stimulus, and simulator configurations are available to perform simulation.

2.1.3 Advantages of Our Methodology

The primary difference between our version of a pruning heuristic and the traditional version is generation of a set of designs as opposed to a single optimal design. The set of designs consists of the designs that meet the given performance constraints. By ensuring that we have a set of good designs as opposed to one optimal design, we increase the chances of finding the optimal design from the set even when approximated high-level models are used. Additional details can be found in [26]. The advantages of hierarchical design space exploration are as follows:

- robust against approximation errors due to high-level abstractions (models) used by the optimization heuristics
- reduces the number of simulations necessary when compared against simulation based design space exploration
- combines the speed of optimization heuristic based DSE and the higher accuracy and parameter coverage of simulation based DSE
- designer can potentially combine different optimization heuristics and high-level estimators to suite the need of target application design problem domain

Low-level detailed simulators, such as SimpleScalar and ModelSim, provide accurate performance estimates but are time consuming. Evaluation of a large design space, even of the order of 10s or 100s, can take days. Our hierarchical design methodology does not require simulation to evaluate the complete design space. Rather, the simulators are used just to estimate performance of different mappings that are used by DESERT and HiPerE to perform DSE. Therefore, our methodology is significantly faster that simulation based DSE. In comparison with optimization heuristic based DSE techniques, due to the use of a high-level estimator, we support higher parameter coverage. Higher parameter coverage results in the evaluation of a larger design space. This is because number of designs in a design space depends on the possible values of various parameters associated with a problem domain. For example, for the problem domain defined by an application and a processor supporting DVS, if the number of discrete voltage settings increases or if we also choose to evaluate a set of memory configurations, the size of the design space will grow.

2.2 Model, Design, and Simulator Reuse in MILAN

MILAN model database stores the models in a canonical form that provides a common representation for the information that are used to drive various simulators. The model database is the basic support for design reuse in MILAN. While designing an application, the designer builds models for the target devices and populates them through simulation results or vendor provided data-sheets. If in the future, another design exercise involved the some of the devices already modeled, the designer can reuse the models. The resource models capture various characteristics of the target devices that can be exploited to measure and optimize performance. They include various operating states, energy dissipation in each state, state transition costs, etc. This information does not change from application to application and hence can be reused. Besides, given an application, it is always modeled as a hierarchical data flow graph of the constituent tasks. The tasks typically are basic signal processing algorithms like matrix multiplication, FFT, matrix decomposition, motion estimation, and are part of many different signal processing applications. Therefore, the task specific information contained in the application model and model for mapping can also be reused. If reused, the designer does not need to perform simulations to populate the mapping model for these tasks. Other form of reuse is through the use of simulators already integrated into MILAN. Simulator integration depends on the underlying modeling paradigm and not on a specific model. Therefore, if the modeling paradigm does not change for a new application design problem, simulators integrated earlier are ready to use.

2.3 MILAN for Reconfigurable Systems

The modeling and performance estimation support for FPGA provided in MILAN is based on Domain Specific Modeling [7]. The focus is on FPGA based designs for typical signal processing algorithms that contain loops and are data oblivious. Matrix multiply, motion estimation, etc. are some such examples. There are numerous ways to map an algorithm onto an FPGA as opposed to mapping onto a traditional processor such as a RISC processor or a DSP, for which the architecture and the components such as ALU, data path, memory, etc. are well defined. For FPGAs, the basic element is the lookup table (LUT), which is too low-level an entity to be considered for high-level modeling. Therefore we use domain specific modeling to facilitate high-level modeling of FPGAs.

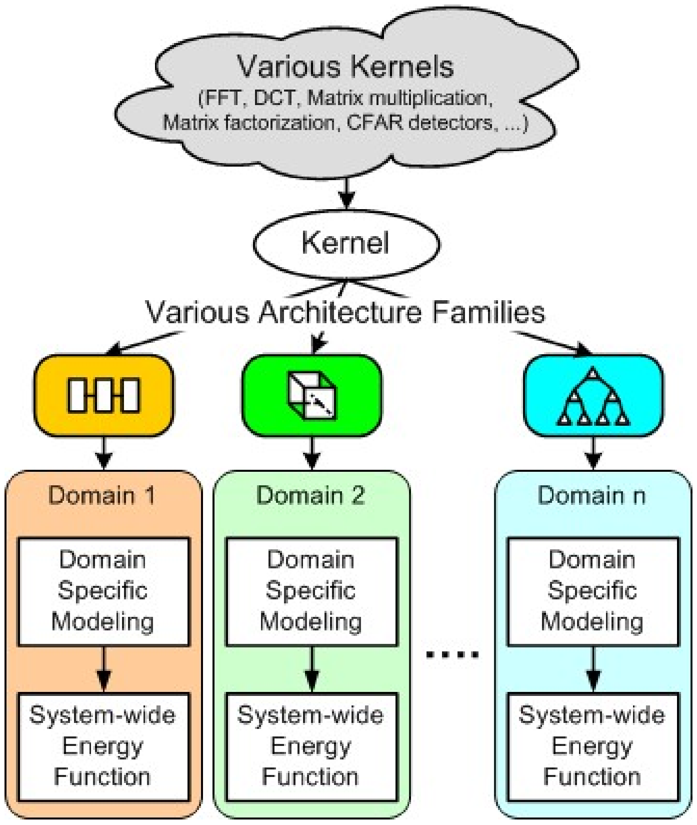
Various Kernels
(FFT, DCT, Matrix multiplication, Matrix factorization, CFAR detectors, ...)
Kernel
Various Architecture Families
Domain 1
Domain Specific Modeling
System-wide Energy Function
Domain 2
Domain Specific Modeling
System-wide Energy Function
....
Domain n
Domain Specific Modeling
System-wide Energy Function

Figure 6: Domain Specific Modeling

Domain-specific modeling technique facilitates high-level energy modeling for a specific domain. A domain corresponds to a family of architectures and algorithms that implements a given kernel. For example, a set of algorithms implementing matrix multiplication on a linear array is a domain. Detailed knowledge of the domain is exploited to identify the architecture parameters for the analysis of the energy dissipation of the resulting designs in the domain. By restricting our modeling to a specific domain, we reduce the number of architecture parameters and their ranges, thereby significantly reducing the design space. A limited number of architecture parameters also facilitate development of power functions that estimate the power dissipated by each component (a building block of a design). For a specific design, the component specific power functions, parameter values associated with the design, and the cycle specific power state of each component is combined to specify a system-wide energy function. More details on domain specific modeling can be found in [7]. We have enhanced MILAN to facilitate modeling of FPGA based designs using the domain specific modeling technique.

2.3.1 Modeling

Our kernel level modeling enables specification of parameterized design for signal processing kernels for implementation using FPGAs. We exploit domain specific modeling, a technique for high-level modeling of FPGAs, developed by Choi et al. [7]. This technique has been demonstrated successfully for designing energy efficient signal processing kernels using FPGAs. A domain refers to a class of architectures and the corresponding algorithms for a particular signal processing kernel. A class of architectures can be a uniprocessor, linear array of processors, 2-D array of processors, or any other class of parameterized architectures. For example, matrix multiplication on a linear array of processors is a domain. A model defined using this technique consists of RModules, Interconnects, component specific parameters and power functions, component power state matrices, and a system-wide energy function. A *Relocatable Module (RModule)* is a high-level architecture abstraction of a computation or storage module. For hardware implementations on an FPGA, a register can be a RModule if the number of registers in the design can vary based on algorithmic level choices. *Interconnect* represents the resources used for data transfer between the RModules. A component (also referred to as building block) can be a RModule or an Interconnect. *Component specific parameters* depend on the characteristics of the component and its relationship to the algorithm. For example, degree of parallelism, precision, size of internal memory (on FPGA), binding options for RModules, power states, are possible component specific parameters. *Component specific power functions* capture the effect of component specific parameters on the average power dissipation of the component. For this we assume a switching activity of 12.5%. *Component Power State (CPS) matrices* capture the power state for all the components in each cycle. For example, consider a design that contains k different types of components ($C_1,\ldots,C_k$) with n_i components of type i. If the design has the latency of T cycles, then k two dimensional matrices are constructed where the i-th matrix is of size $T\times n_i$ (Figure 6). An entry in a CPS matrix represents the power state (e.g. active or clock-gated) of a component during a specific cycle and is determined by the algorithm. *System-wide energy function* represents the energy dissipation of the designs belonging to a specific domain as a function of the parameters associated with the domain.

Modeling based on the technique described above has the following advantages.

- various parameters get exposed at the algorithm level
- performance models for energy, area, and latency are generated in the form of parameterized functions
- it is possible to rapidly estimate different performance metrics using only the information captured in the models
- a parameterized model of a domain captures a set of designs (based on parameter values) that can be analyzed for various performance tradeoffs.

We provide a hierarchical modeling support to model the datapath. The hierarchy consists of three types of components; micro, macro, and basic blocks. A basic block is target FPGA specific. For example, the basic blocks specific to Xilinx Virtex II Pro are LUT, embedded memory cell, I/O Pad, embedded multiplier, and interconnects. In contrast, for Actel ProASIC 500 series of devices, there will be no embedded multiplier. Micro blocks are basic architecture components such as adders, counters, multiplexers, etc. designed using the basic blocks. In principle, there is no difference between a basic block and a micro block. The classification is introduced to enable logical creation of a basic library per device. A macro block is an architecture component that is used by some instance of the target class of architectures associated with the domain. For example, if linear array of processing elements (PE) is our target architecture, a PE is a macro block.

Each building block is associated with a set of component specific parameters. Power states is one such parameter, which refers to various operating states of each building block. For example, we can model two states, ON and OFF for each micro and basic block. In ON state the component is active and in OFF state it is clock gated. For macro blocks it is possible to have more than 2 states due to different combination of states of the constituent micro and basic blocks. Power is specified as a function or constant value (in the example model, power for different components are specified as constants).

In addition each block can be associated with a set of variables. Precision, depth and width for memory, size of register or memory are some example of variables that can be associated with a component.

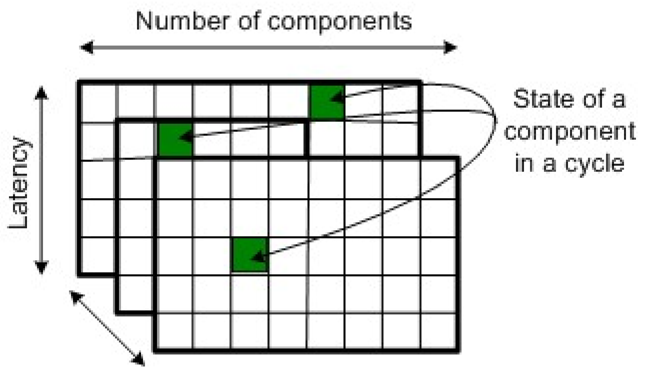

Figure 7: CPS matrices

While the datapath is modeled as specified above, the model for control flow is relatively tricky. Our focus of the modeling and estimation capability is rapid energy, latency, and area estimation. Area can be estimated based on the model of the data path (sum of the components' areas). In order to model the control flow we make use of CPS matrices. Component Power State (CPS) matrices capture the power state for all the components in each cycle. For example, consider a design that contains k different types of components (C_1,...,C_k) with n_i components of type i. If the design has the latency of T cycles, then k two dimensional matrices are constructed where the i-th matrix is of size Txn_i. An entry in a CPS matrix represents the power state of a component during a specific cycle and is determined by the algorithm (Figure 6).

However, specification of such a matrix is not easy. Hence, we take advantage of the typical loop oriented structures of kernels such as matrix multiply, FFT, etc. for which the FPGA based designs are created. If we analyze the CPS matrices, we can observe that another easy way to specify the same information is through a table. Such table would contain a number of rows where each row is a 3-tuple (component, state, #of cycles in this state). As we are interested only in performance estimation, this much of information is enough.

2.3.2 Design Flow for Reconfigurable Systems

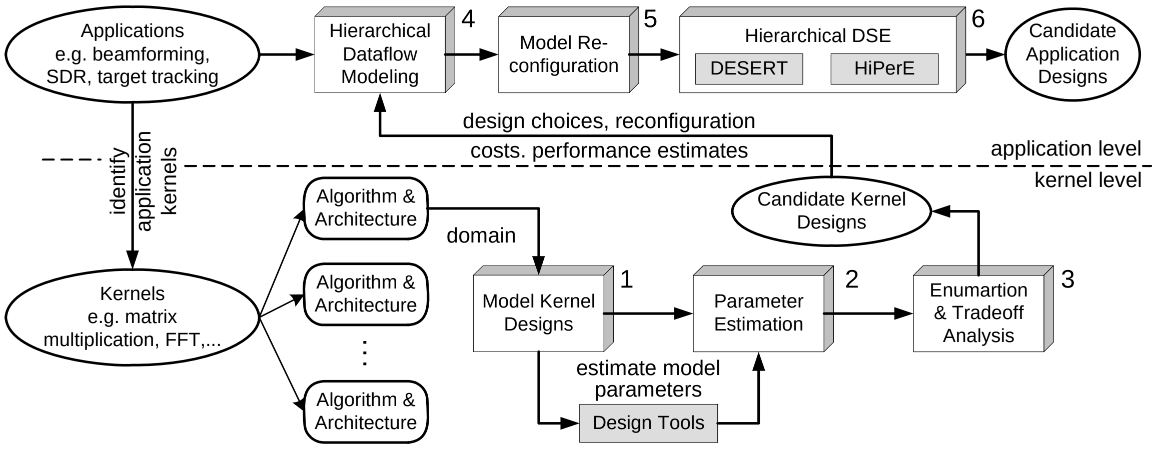
Applications e.g. beamforming, SDR, target tracking
Hierarchical Dataflow Modeling
4
Model Re-configuration
5
Hierarchical DSE
DESERT
HiPerE
6
Candidate Application Designs
design choices, reconfiguration
costs. performance estimates
application level
kernel level
identify application kernels
Candidate Kernel Designs
Algorithm & Architecture
domain
Kernels e.g. matrix multiplication, FFT,...
Algorithm & Architecture
Model Kernel Designs
1
Parameter Estimation
2
Enumartion & Tradeoff Analysis
3
Algorithm & Architecture
estimate model parameters
Design Tools

Figure 8: Design flow for Reconfigurable Systems

As shown in Figure 7, design flow using our framework consists of 6 steps. The first three steps deal with modeling the kernel designs based on domain specific modeling and identifying design choices. The last 3 steps perform application level modeling and design space exploration (These steps are also described in the previous chapter). In the following, we discuss each step in detail.

- Modeling Kernel Design (1): In this step, the designer analyzes the kernels to define domain specific models. The designer identifies the micro, macro, and library blocks and the associated component specific parameters. The model of the data path is graphically constructed in this step using GME. The designer can also specify high-level scripts for the building blocks to be used in the next step. In addition, CPS matrices for the algorithm are also specified.
- Parameter Estimation (2): Estimation of the cost functions for power and area involves synthesis of a building block, low-level simulations, and in case of power, the use of confidence intervals to generate statistically significant power estimates. The simulations are performed off-line or, if required simulator is integrated, automatically using specified high-level scripts. Instead, if a library of models is available, the stored performance estimates are used directly. Latency functions are estimated using the CPS matrices. System-wide energy is estimated using the latency function and component specific power functions.

- Enumeration and Tradeoff Analysis (3): In this step, the designer chooses the candidate kernel designs that would be evaluated while designing applications. Given a domain specific model of a kernel, a set of designs are identified based on the parameter values and binding choices. The framework also generates comparison graphs to compare the performance of the designs.
- Hierarchical Data flow Modeling (4): Once, we have identified implementation choices for each kernel, we construct the application model as a hierarchical data flow with alternatives. Compound, alternative, and leaf nodes are used to specify the application model. The leaf nodes are also associated with FPGAs on which the kernel will be implemented. In addition, each leaf node is associated with performance estimates obtained using the high-level performance estimator.
- Modeling Reconfiguration (5): Based on the mapping and area estimates of the task implementations, pseudo nodes are introduced to model reconfiguration. This step is automatic within our framework. The application model is analyzed using topological sort and for each consecutive tasks (source and destination) executing on a single FPGA, the application model introduces a pseudo task. Each pseudo task is automatically associated with a set of alternatives and design constraints are introduced to ensure that correct reconfiguration is chosen based on the choices selected for the source and destination tasks.
- Hierarchical DSE (6): This step uses DESERT and HiPerE to explore the design space using the application model. DESERT applies all the performance and design constraints and selects a set of designs that meet the constraints. HiPerE evaluates the selected designs based on their performance estimates and allows the designer to choose the final design based on the given performance requirements. In the following, we discuss this step in detail.

2.3.3 Illustrative Design Space Exploration for Reconfigurable Devices

A matrix multiplication algorithm for linear array architectures is proposed in [32]. We use this algorithm to demonstrate modeling, high-level performance estimation, and performance tradeoff analysis capabilities of the design framework. Thus it uses only Step 1, 2, and 3 of the design flow. The focus is to generate a set of energy efficient designs for matrix multiply using Xilinx Virtex-II Pro.

In Step 1, the architecture and the algorithm were analyzed to define the domain specific model. Various building blocks that were identified are register, multiplexer, multiplier, adder, processing element (PE), and interconnects between the PEs. Among these building blocks only the PE is a library block and the rest of the components are micro blocks. Component specific parameters for the PE include number of register (s) and power states ON and OFF. ON refers to the state when the multiplier (within the PE) is in ON state and OFF refers to the state when the multiplier is in OFF state. Additionally, for the complete kernel design number of PEs (pe) is also a parameter. For $N \times N$ matrix multiplication, the range of values for s is $1 \le s \le N$ and for pe it is $1 \le pe \le N(\lceil N/s \rceil.)$ For matrix multiplication with larger size matrices (large values of N) it is not possible to synthesize the required number of PEs due to area constraint. In such cases, block matrix multiplication is used. Therefore, block-size (bs) is also a parameter.

Once the data path was modeled we generated the cost function for power and area for the different components. Switching activity was the only parameter for power functions. To define the CPS matrices, we analyzed the algorithm to identify the operating state of each component in different cycles. As per the algorithm [12], in each PE, the multiplier is in ON state for $T/(\lceil n/s \rceil)$ cycles and is in OFF state for $T\times(1-1/\lceil n/s \rceil)$) cycles. All other components are active for the complete duration.

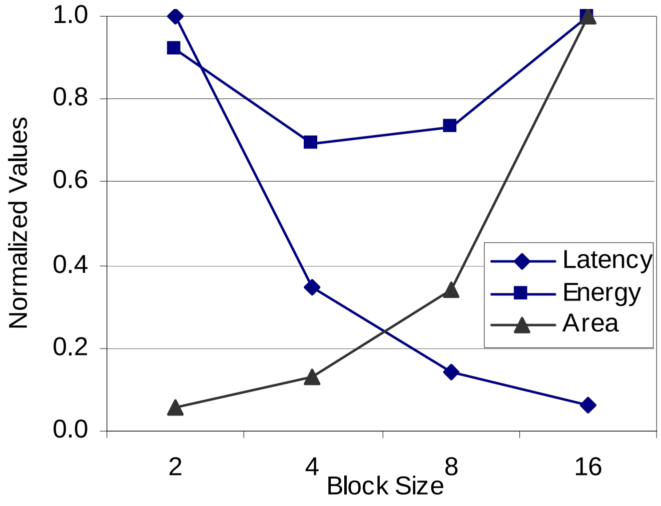

Figure 9: Energy, latency, and area tradeoffs

In Step 2, we performed simulations to estimate the power dissipated and area occupied by the building blocks. The latency (T) of this design using $N\lceil N/s \rceil$ PEs and s storage per PE is $T=(N^2+2N(\lceil N/s \rceil) - (\lceil N/s \rceil) + 1)$. Using the latency function, component specific power functions, and CPS matrices, we derived the system-wide energy function.

Finally, we analyzed the model to identify a set of designs that provide a tradeoff between different performance metrics. Figure 8 shows the variation of energy, latency, and area for different block sizes for 16×16 matrix multiplication. It can be observed that energy is minimum at a block size of 4 and area and latency are minimum at block size 2 and 16 respectively. This information is used to identify a suitable design (block size) based on latency, energy, or area requirements.

Figure 9 shows energy distribution among multipliers, registers, and I/O pads for three different designs. Design 1 corresponds to the original design described in [32] and Design 2 and 3 are low energy variants discussed in [12]. Using Figure 9, we identify that the registers dissipate the maximum energy and select them as candidates for optimization. Optimizations considered include reduction of number of registers through analysis of data movements (Design 2) and use of CLB based SRAMs instead of registers to reduce energy dissipation (Design 3). Details of the optimized algorithm are available in [12].

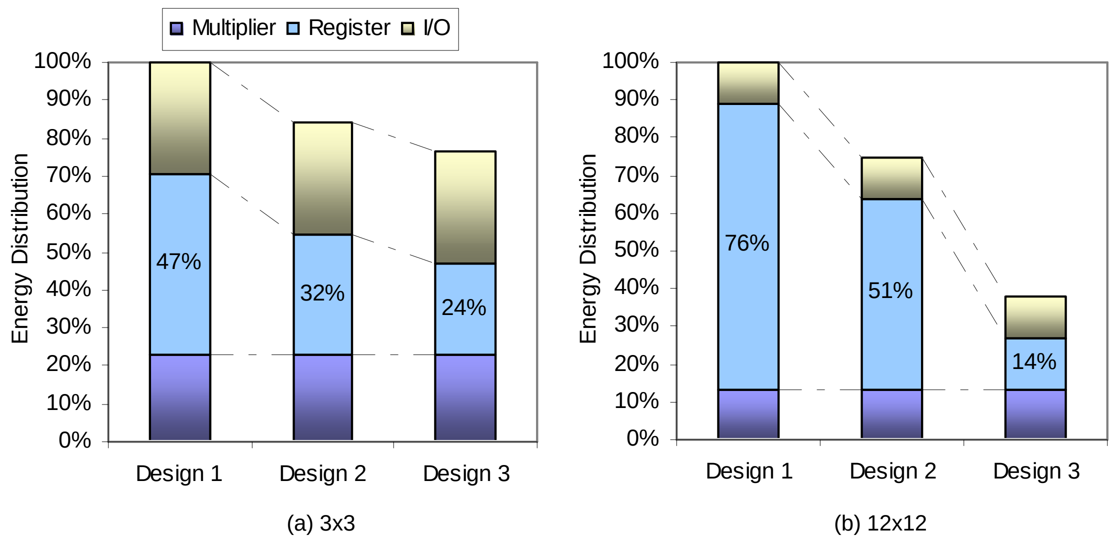

Figure 10: Reducing energy dissipation

2.4 Multigranular Simulation

One important requirement of the heterogeneous design paradigm is the orthogonalization of concerns, that is to separate various aspects of design in order to effectively explore alternative solutions. For example, system requirement specifications and implementation or computation and communication are good candidate concerns that should be separated. In large and complex systems there is a need for modular design to mitigate complexity. Systems are typically designed in terms of components and component interactions. A component usually embodies some kind of computation and it has a standardized interface for communication. This helps to separate computation from communication and the developer can design and implement one without being concerned with the other.

Separation of system requirements and implementation is desirable because the former captures the intention of the system designer and provide a high level view, while the latter is specific and is done at a much finer level of granularity. By capturing the intention separate of the implementation, the high level abstraction is preserved, allowing the user to specify alternate implementations for the same intent. These alternatives may be in the form of different algorithms to solve the same problem, a choice between hardware and software implementation, or a selection of programming language. Furthermore, implementation is a refinement of the intent and needs to be captured at different levels of granularity. Initially a coarse grain implementation is used for prototyping. This can be transformed in stages to a detailed low-level implementation later.

By capturing alternative implementations at different levels of granularity we gain the flexibility of choosing the implementation according to the exact needs of the system. The development cycle starts from a coarse grain implementation. This is tested for functional correctness and is then refined to different alternative implementations. The feasibility of these alternatives is explored by profiling them. This is followed by system simulation of a few feasible system wide implementations to validate the system with respect to the requirements. Simulation becomes more important as testing of these applications on actual hardware is expensive and time consuming, especially for applications implemented in hardware such as FPGAs or ASICs.

The MILAN modeling language incorporates a wide variety of domain specific modeling concepts that as follows:

- A separate modeling sublanguage for modeling of hardware applications that uses domain specific concepts such as hardware modules, ports, busses clocks and event triggered functions,
- Strong data typing of communication ports for accurate simulation of data exchange and to catch modeling errors at design time,
- Parameterization of components to develop generic modules for reuse, as well as to design a set of solutions instead of a single solution,
- Data abstraction and information hiding to better manage complexity using multiple aspects of the same module,
- Explicit designs of alternative implementations to capture design choices in order to better explore different solutions, and
- A paradigm to compose hardware and software components together to facilitate the design of heterogeneous systems.

Hierarchy in the modeling paradigm serves two purposes. First, it helps separate intention from implementation. Using hierarchy the system is designed according to the intention, that is, the high-level dataflow of the system. Then the design in refined by designing the modules in detail until it is low-level enough to provide an implementation. Second, it helps to mitigate complexity. The dataflow graph of large systems can be very complex; hierarchy hides data at different levels to make the systems more manageable.

In the hierarchical graph representing the system, typically the lowest level modules contain the behavioral information. Using multi-granular simulation the user can choose to provide behavioral information for any module at any level of hierarchy. Thus, the user can simulate a system with a mixture of coarse grain and fine grain implementations. The code generator synthesizes the code for the system and whenever it finds a module marked for using the coarse grain implementation it uses that and doesn’t traverse deeper in that module.

2.5 Accurate Functional Simulation

There are two kinds of simulations, performance and functional, that are performed on a system under development. Performance simulators simulate the system to generate the metrics for system parameters such as latency, throughput, power, energy and so on, while the functional simulator simulates the system’s functionality. Designing an embedded system typically involves performing functional simulation on it before committing the design. A functional simulation essentially verifies the behavior of a system at a higher-level and helps capture the design errors and inconsistencies at an earlier stage before proceeding to a detailed implementation of the system. It lets the designer verify a coarse-grain implementation of the system with its functional requirements.

Accurate functional simulation of a system based on the dataflow approach that MILAN advocates depends on two factors: accurate functional representation of the dataflow components and the correct simulation of the runtime scheduler (in case of asynchronous dataflow). Providing functional code for the dataflow components is the users' job. However, MILAN needs to schedule the dataflow graph exactly as the real runtime system does.

MILAN supports three functional simulators: Matlab, SystemC and VHDL. The MILAN framework is capable of simulating both synchronous and asynchronous dataflow. Synchronous dataflow can be scheduled statically in compile time. MILAN uses the same code generator module for both simulation and system synthesis ensuring consistency. MILAN includes a MATLAB component that implements the same scheduling strategies as the runtime system. Both a simple round robin scheduling strategy and several priority based schemes are supported. The MATLAB environment provides the same simulation results as the final target system for the same models and the same conditions provided the component implementation are identical.

The design of functionality implemented in hardware is captured using an intuitive graphical language supported by MILAN. The entire system's description in VHDL and/or SystemC is automatically generated from them using a model-interpreter. It also facilitates the automatic generation of data type package, glue code, clock code, and parameters from the application models. Furthermore, the generated VHDL descriptions are in conformance with the IEEE Std 1076-1993 standard of the VHDL. Any simulator compatible with this standard can be used to compile the generated code and simulate it.

Furthermore, different simulation modes are also supported. In order to simulate a module in isolation, the module needs to be driven by sourcing functions and the output of the module needs to be sent to sinking functions. In MILAN we allow the user to capture the exact sourcing and sinking function associated with each communication port. Hence, to synthesize code for an isolated simulation of a module, the true implementation of the module in question is used along with the sourcing and sinking functions from adjacent modules. The interpreter generates code of the module in question and creates sourcing and sinking modules for it. Isolated simulation can be performed not only on a single module, but also on a subgraph and the modules adjacent to this graph will be used to supply and consume data.

For a complete simulation of the entire system a single design needs to be chosen. The user can choose between alternative implementations by marking one of various alternative implementations to use. Alternatively, the design-space exploration tool can identify the point designs that satisfy the all constraints and mark the selected alternatives automatically. The interpreter then traverses through the models and picks up the chosen alternative implementations to form a single design. The true implementations of the design are then used to generate Matlab, VHDL or SystemC code for a full simulation.

To simulate hardware in a heterogeneous system, it is necessary to facilitate communication between hardware and software components. In a real-world system, hardware-software interactions are facilitated using device drivers. However, MILAN does not require device drivers to simulate the system. The communication is achieved by using entities called proxies. At a hardware-software interface, proxies are generated on both sides. For example, a hardware proxy will read data from the hardware module at the interface and pipe it to its software counterpart using TCP. Similarly, it will read data from the pipe and provide it to the hardware module. The software proxy does the same at the other end. The interpreter breaks the heterogeneous graph into hardware and software graphs. It then generates the proxies and connects the respective graphs at the interface. Finally, the two graphs are sent through their respective interpreters. The hardware and software code can finally be compiled independently and then run together to simulate the hardware.

3 Lessons Learned

The MILAN project studied several key phases required for energy and latency efficient application design using embedded systems. We discuss a number of lessons learned in this chapter.

3.1 Application modeling

Synchronous data flow (SDF) emerged as a simple yet powerful abstraction to model signal processing applications especially for energy efficient design using power aware embedded systems. SDF specifies an application as a directed acyclic graph where the nodes model the application tasks and the edges model the order of execution among the tasks. SDF allows efficient scheduling of application tasks via topological sorting of the task graph while exploiting available parallelism. In order to support low-power design our target processing and memory devices supported a number of operating states. Therefore, during application design, it is required to identify the most suitable operating state for an application task while meeting given performance constraints. This requires operating state transition between successive task executions. SDF emerged as a suitable model that was easily extended to model operating state transition. SDF was also well suited for design space exploration using DESERT, performance evaluation using HiPerE, and automatic high-level code generation. For the MILAN design environment, SDF was also extended easily to support a hierarchical modeling that allowed user-friendly graphical interface towards application modeling. A variant of SDF, asynchronous data flow (ASDF) was also supported by MILAN. Both SDF and ASDF combined have proved to be simple yet powerful abstractions to model a wide variety of signal processing applications especially used in the defense community.

3.2 Simulator integration

The main focus of the MILAN project was to develop a design environment that allowed integration of a wide variety of popular simulators suitable for embedded devices. A number of such simulators are available from academia and industry. Some examples are SimpleScalar, SimplePower, PowerAnalyzer, JouleTrack, ARMulator, Xilinx XPower, ModelSim, among others. The biggest concern while integrating these simulators is lack of a standard interface and difference in simulation speed. Therefore, it is a very difficult task to integrate these simulators in a manner that they interact with each other to estimate system-wide performance for a heterogeneous embedded system that integrates several processing and memory devices. Therefore, MILAN project has developed a hierarchical simulation technique [26]. Hierarchical simulation relies on a high-level performance estimator (HiPerE) to generate system-wide performance estimates by integrating components specific performance estimates generated using the aforementioned simulators. Thus HiPerE hides the lack of common interface and difference in speed among the low-level simulators. MILAN, through the use of GME, allows integration of individual simulators that are driven by the models supported by MILAN. Thus, given appropriate high-level code and input stimuli for an application task, simulation can be automatically performed and the component specific performance estimate can be automatically updated in the MILAN models. HiPerE uses the updated models to generate system-wide performance estimates. Use of HiPerE also allows reuse of component specific performance estimates.

3.3 Device selection

The MILAN project studied the problem of device selection. This problem assumes that the target application is specified but the target hardware is not specified. Instead a number of processing and memory devices are available which should be evaluated to identify a suitable combination of devices that meet the functional and performance requirements. Other aspects that can be considered are size, cost, and weight. Device selection is a larger problem (with respect to design space) than application mapping and scheduling as device selection includes mapping and scheduling while evaluating a number of devices, which significantly increases the design space. MILAN is able to evaluate a given set of devices due to its support for simulator integration (for each candidate device) and availability of HiPerE. Device selection also requires design constraints in addition to performance constraints. Design constraints specify valid combination of devices. For example, a designer may not want to choose a target heterogeneous embedded system, which integrates 2 FPGAs but would want 1 FPGA with a general purpose processor. MILAN supports object constraint language (OCL). OCL was identified as an efficient technique to specify design constraints and our design space exploration tool, DESERT, could support design space exploration based on design constraints without any modification.

3.4 Hierarchical design space exploration

MILAN supports hierarchical design space exploration which integrates a pruning heuristic followed high-level estimation and low-level simulation. The primary difference between our version of a pruning heuristic and the traditional version of optimization heuristics is generation of a set of designs as opposed to a single optimal design. The set of designs consists of the designs that meet the given performance constraints. By ensuring that we have a set of good designs as opposed to one optimal design, we increase the chances of finding the *real-optimal* design from the set even when approximated high-level models are used. An optimal design is the best design identified (based on the performance requirements) by the optimization heuristic using the underlying approximated high-level model. A real-optimal design is the design that is optimal when the designs are implemented using hardware and performance is measured. A real-optimal design can be different than the design identified by the optimization heuristic because the later assumes a high-level approximated model with lower parameter coverage. For ease of comparison, we assume that the most detailed low-level simulator available is accurate and can be used to identify the real optimal solution. We also assume that the errors induced by approximations are marginally low when compared with the actual performance values. Hierarchical design space exploration is robust against approximation errors due to high-level abstractions (models) used by the optimization heuristics and reduces the number of simulations necessary when compared against simulation based design space exploration. Hierarchical design space exploration allows a designer to potentially combine different pruning heuristics and high-level estimators to suite the need of target application design problem domain.

3.5 Duty cycle based design space exploration

Duty cycle is the proportion of time during which a system is operated. Such specification allows modeling of a period of execution as alternate active and inactive phases. Energy dissipation (e.g. due to leakage current), especially for systems with low duty cycle, during the inactive phases can contribute significantly to the overall energy dissipation of the system. Therefore, the tradeoff between the performance cost of shutting down and starting up a device and the performance cost of remaining idle needs to be considered during system design. HiPerE, a high-level performance estimator integrated in the MILAN framework allows performance estimation and design space exploration based on duty cycle specification.

4 Technology Transition

The MILAN environment has been used for several different projects. By the time this document was produced, there have been approximately 240 unique downloads (see Appendix) of the MILAN framework. We briefly discuss two projects in this section.

4.1 Power Aware Remote Information System (PARIS)

The focus of the PARIS project is to identify an energy efficient implementation of a personnel detection algorithm. The personnel detection algorithm is required to processes input in real-time and hence there is a hard latency requirement. In addition, as the system needs to be deployed in a power-constrained environment, energy dissipation is also an important metric. We used MILAN to identify an energy efficient hardware and the corresponding mapping for the above algorithm from a set of devices that consists of traditional processors, FPGAs, and DSPs.

Using MILAN, we modeled the application (a 5 stage linear array of tasks) and the candidate hardware choices. The PARIS application is a multi-rate application like the beamforming algorithm discussed earlier. Resource modeling for PARIS involved modeling of different operating states, performance cost of state transitions, and power consumption for each state. We also performed several simulations to estimate the performance values associated with the mappings of the application tasks and the target hardware devices. PARIS project allows only certain combination of devices. These design constraints were specified using OCL. We also evaluated both floating and fixed-point implementation for each application task. The size of the design space prior to the use of DESERT was approximately 73,000.

Following modeling, we performed a two-level design space exploration using DESERT and HiPerE. DESERT cannot evaluate designs based on duty-cycle specifications. However, it is not practical to use simulation to evaluate 73,000 designs. For example, simulation of a design using TI Code Composer Studio (for DSP) takes approximately 25-30 minutes. Even with HiPerE, it takes approximately 10 hours to estimate the performance of all the designs and a tedious manual comparison of all the estimates to identify the energy efficient design. Therefore, DESERT is used to evaluate the large design space based on the latency requirement. We initially assumed a constant rate application model and evaluated all the target hardware for a single instance of execution of the application. Later, we used HiPerE to evaluate the designs identified by DESERT. Evaluation of the designs using HiPerE was based on duty-cycle specification and the design with minimum energy dissipation was selected as the final design and the associated architecture was identified as the target architecture.

Initially, DESERT identified two candidate architectures, Virtex-II Pro and a combination of Actel ProASIC and TI DSP. Virtex-II Pro based designs were more efficient in terms of latency and energy both for a single instance of execution. However, based on duty-cycle specifications (which evaluates a design over a period of time that includes, data processing and idling or shut-down and start-up), the combination of Actel

ProASIC and DSP turned out to be more efficient. This is because Virtex-II Pro has a very high quiescent power and start-up cost.

4.2 Power Aware Sensing and Tracking Analysis (PASTA)

The PASTA application design problem is to identify an energy efficient mapping of a automated target recognition (ATR) application onto a heterogeneous embedded system while meeting the given latency constraint. The underlying architecture for the PASTA project is already specified (http://pasta.east.isis.edu).

The hardware includes sensor(s), a processor, several microcontrollers, memories, and a radio. Each component can be independently turned on or off. In addition, the processor (Intel PXA 255) supports voltage and frequency scaling. The target application is an automated target recognition algorithm that performs beamforming based on acoustic signals from the sensors. The beamforming application consists of a linear array of 6 tasks. The first three tasks are “receive data” which is mapped onto the radio, “sampling” which is mapped onto the microcontroller, and “false-alarm detection” which can be mapped onto either the microcontroller or the processor. The last three tasks that compute beamforming are FFT, peak-pick, and delay sum.

The design problem for PASTA involves identification of the operating state of each component for each task such that the complete ATR application dissipates the minimum energy while satisfying the latency requirement. All the components in the PASTA stack have at least two operating states; ON and OFF. There is a constant amount of time and energy spent to switch on each component. In addition, the processor has 6 different operating frequencies. Tasks can be mapped onto the processor or the microcontroller. When mapped onto a processor, the task can be executed in a certain operating state and the performance of the mapping depends on the operating state. Transition between any two operating states also involves time and energy costs which depend on the source and destination states. We modeled all the above in MILAN. The resulting design space was approximately 500,000. However, we noticed that the transition costs between different operating states of the processor are negligible except one transition, which involves changing operating frequency of the bus. Hence, the design space was reduced to 12,000. As with the other examples, we used simulators for Intel PXA 255 and the microcontroller to estimate performance of all the mappings. The start-up costs and state transition costs are estimated based on the data sheets provided by the vendors.

Design space exploration involved DESERT and HiPerE. For design space exploration, the latency constraint was assumed to be < 1 sec. DESERT initially pruned the design space to 10 designs based on the latency constraint. DESERT does not include the performance of the sensor, radio, and the memory while evaluating the design space. Hence we used HiPerE to identify the design with minimum energy dissipation. In the resulting design, the components, which are idle, are switched off. For example, while the radio is receiving data, the processor is turned off and is turned on only when data is ready for processing. The design also maps the false-alarm detection task onto the microcontroller, as while latency is higher compared with the PXA processor, energy dissipation is lower.

5 Publications Acknowledging this Contract

The list of documents and published manuscripts attached to the final report are described below. The “File Name” corresponds to the name of the soft copy file for each document.

Num	Title
1	MILAN User Manual 1.1
2	MILAN: A Model Based Integrated Simulation Framework for Design of Embedded Systems, Agrawal A., Bakshi A., Davis J., Eames B., Ledeczi A., Mohanty S., Mathur V., Neema S., Nordstrom G., Prasanna V., Raghavendra, C., Singh M. Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES 2001), Snowbird, Utah, June 2001
3	A Hierarchical Simulation Framework for Application Development on System-on-Chip Architectures, Mathur V. and Prasanna V. K., 14th IEEE Int'l ASIC-SOC Conference, Washington DC, September 2001
4	Rapid Design Space Exploration of Heterogeneous Embedded Systems using Symbolic Search and Multi-Granular Simulation, Sumit Mohanty, Viktor K. Prasanna, Sandeep Neema, and James Davis, Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES 2002), Berlin, Germany, June 2002
5	Domain-Specific Modeling for Rapid System-Wide Energy Estimation of Reconfigurable Architectures, Seonil Choi, Ju-wook Jang, Sumit Mohanty, Viktor K. Prasanna, The International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA 2002), Las Vegas, Nevada, USA
6	A Model-based Methodology for Application Specific Energy Efficient Data Path Design using FPGAs, Sumit Mohanty, Seonil Choi, Ju-wook Jang, Viktor K. Prasanna, IEEE 13th International Conference on Application-specific Systems, Architectures and Processors (ASAP 2002), San Jose, California
7	Rapid System-Level Performance Evaluation and Optimization for Application Mapping onto SoC Architectures, Sumit Mohanty and Viktor K. Prasanna, 15th IEEE International ASIC/SOC Conference, Rochester, New York
8	Towards Automatic Synthesis of a Class of Sensor Network Applications, Amol Bakshi, Jingzhao Ou and Viktor K. Prasanna, Intl. Conf. On Compilers, Architectures, and Synthesis for Embedded Systems (CASES), October 2002
9	Ledeczi A., Davis J., Neema S., Agrawal A.: Modeling Methodology for Integrated Simulation of Embedded Systems, ACM Transactions on Modeling and Computer Simulation, 13, 1, pp. 82-103, January, 2003
10	Agrawal A., Ledeczi A.: Multigranular Simulation of Heterogeneous Embedded Systems, Tenth IEEE Conference and Workshops on the Engineering of Computer Based Systems (ECBS), p. 3-10, Huntsville, Alabama, April 7, 2003.
11	An Algorithm Designer's Workbench for Platform FPGAs, Sumit Mohanty and Viktor K Prasanna, 13th International Conference on Field Programmable Logic and Applications (FPL 2003), September 2003
12	A Modeling and Exploration Framework for Mapping of Linear Array of Tasks onto Adaptive Computing Systems, Egor Andreev, Sumit Mohanty, and Viktor K Prasanna, 6th Annual Military and Aerospace Programmable Logic Devices (MAPLD) International Conference, September 2003
13	A Hierarchical Approach for Energy Efficient Application Design Using Heterogeneous

	Embedded Systems, Sumit Mohanty and Viktor K Prasanna, International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES 2003), October-November 2003
14	Domain-Specific Modeling for Rapid Energy Estimation of Reconfigurable Architectures, Seonil Choi, Ju-wook Jang, Sumit Mohanty, and Viktor K. Prasanna, Special Issue on Configurable Computing of the Journal of Supercomputing, Kluwer
15	MILAN: A Design Environment for Latency and Energy Efficient Implementation of Adaptive Antenna Applications, Sumit Mohanty, Jingzhao Ou, and Viktor K. Prasanna, to appear in Adaptive Antenna Arrays: Trends and Applications, Ed. Satish Chandran, Springer, 2004
16	Design of High-Performance Embedded System using Model Integrated Computing, Sumit Mohanty and Viktor K. Prasanna, 2nd RTAS Workshop on Model-Driven Embedded Systems, 2004
17	A Framework for Energy Efficient Design of Multi-Rate Applications using Hybrid Reconfigurable System, Sumit Mohanty and Viktor K. Prasanna, Field Programmable Logic and its Application (FPL), 2004

6 References and Related Publications

[1] Actel ProASIC Plus. http://www.actel.com/products/proasic/

[2] Agrawal A..: “Hardware Modeling and Simulation of Embedded Applications”, Master's Thesis, Vanderbilt University, May, 2002.

[3] Agrawal A., Bakshi A., Davis J., Eames B., Ledeczi A., Mohanty S., Mathur V., Neema S., Nordstrom G., Prasanna V. K., Raghavendra C., Singh M., “MILAN: A Model Based Integrated Simulation for Design of Embedded Systems,” Language Compilers and Tools for Embedded Systems, 2001.

[4] ARMulator. http://www.arm.com/support/ARMulator.html

[5] Bakshi A., Ou J., and Prasanna V. K., “Towards Automatic Synthesis of a Class of Application-Specific Sensor Networks,” Intl. Conf. on Compilers, Architecture, and Synthesis for Embedded System, 2002.

[6] Bohrer P, et. al: “Mambo -- A Full System Simulator for the PowerPC Architecture,” ACM SIGMETRICS Performance Evaluation Review, 31(4): 8-12, March 2004.

[7] Choi S., Jang J., Mohanty S., and Prasanna V K, “Domain-Specific Modeling for Rapid System-Wide Energy Estimation of Reconfigurable Architectures,” Engineering of Reconfigurable Systems and Algorithms, 2002.

[8] Farkas, J.: “Asynchronous dataflow scheduling in the MATLAB environment”, M.S. Thesis, Vanderbilt University, 2002.

[9] Generic Modeling Environment. http://www.isis.vanderbilt.edu/Projects/gme/

[10] IBM PowerPC 405. http://www-3.ibm.com/chips/techlib/techlib.nsf/products/PowerPC_405_Embedded_Cores

[11] Intel PXA 255/270. http://www.intel.com/design/pca/prodbref/252780.htm

[12] Jang J., Choi S. and Prasanna V. K.: “Energy-Efficient Matrix Multiplication on FPGAs,” Proc. of Field Programmable Logic and Applications, 2002.

[13] JouleTrack: A web based software energy profiling tool. http://www-mtl.mit.edu/research/anantha/jouletrack/JouleTrack/

[14] Karsai G., Sztipanovits J., Ledeczi A., and Bapty T.: Model-Integrated Development of Embedded Software, Proceedings of the IEEE, Vol. 91, Number 1, pp. 145-164, January, 2003.

[15] Lahiri K., Raghunathan A., and Dey S.: “Efficient Power Profiling for Battery-driven Embedded System Design,” IEEE Tran. on Computer-Aided Design of Integrated Circuits and Systems, 2004.

[16] Ledeczi A., et.al.: “GME Users Manual”, available from www.isis.vanderbilt.edu/projects/gme.

[17] Ledeczi A., et.al.: “Composing Domain-Specific Design Environments”, Computer, pp. 44-51, November, 2001.

[18] Ledeczi A., Davis J., Neema S., and Agrawal A.: “Modeling Methodology for Integrated Simulation of Embedded Systems”, ACM Transactions on Modeling and Computer Simulation, 13, 1, pp. 82-103, January, 2003.

[19] Lee E. A. and Messerschmidt D. G.: “Static scheduling of synchronous data flow programs for digital signal processing”, *Transactions on Computers*, C36 (1), 24-35, 1987.

[20] Mathur V. and Prasanna V. K., “A Hierarchical Simulation Framework for Application Development on System-on-Chip Architectures,” IEEE Intl. ASIC/SOC Conference, 2001.

[21] Micron, Mobile SDRAM. http://www.micron.com/

[22] Mohanty S. and Prasanna V. K.: “Rapid System-Level Performance Evaluation and Optimization for Application Mapping onto SoC Architectures,” 15th IEEE Intl. ASIC/SOC Conference, 2002.

[23] Mohanty S, Prasanna V K, Neema S, Davis J: “Rapid Design Space Exploration of Heterogeneous Embedded Systems using Symbolic Search and Multi-Granular Simulation,” Language Compilers and Tools for Embedded Systems, 2002.

[24] Mohanty S. and Prasanna V. K.: “An Algorithm Designer's Workbench for Platform FPGAs,” Field Programmable Logic and Applications, 2003.

[25] Mohanty S., Ou J., and Prasanna V. K.: “An Estimation and Simulation Framework for Energy Efficient Design using Platform FPGAs,” IEEE Symposium on Field-programmable Custom Computing Machine, 2003.

[26] Mohanty S. and Prasanna V K: “A Hierarchical Approach for Energy Efficient Application Design Using Heterogeneous Embedded Systems,” Intl. Conf. on Compilers, Architecture, and Synthesis for Embedded System, 2003.

[27] Mohanty S. and Prasanna V K: “Design of High-Performance Embedded System using Model Integrated Computing,” 2nd RTAS Workshop on Model-Driven Embedded Systems, 2004.

[28] Mohanty S. and Prasanna V K: “A Framework for Energy Efficient Design of Multi-Rate Applications using Hybrid Reconfigurable System,” Field Programmable Logic and its Application, 2004.

[29] Neema S.: “System Level Synthesis of Adaptive Computing Systems,” Doctorate Thesis, Vanderbilt University, Department of Electrical and Computer Engineering, May, 2001.

[30] Ou J., Choi S., and Prasanna V. K.: “Performance Modeling of Reconfigurable SoC Architectures and Energy-Efficient Mapping of a Class of Applications,” Field-Programmable Custom Computing Machines, 2003.

[31] PowerAnalyzer. The SimpleScalar-Arm Power Modeling Project. http://www.eecs.umich.edu/~jringenb/power/

[32] Prasanna V. K. and Tsai Y.: "On Synthesizing Optimal Family of Linear Systolic Arrays for Matrix Multiplication," IEEE Transactions on Computers, Vol. 40, No. 6, 1991.

[33] Riley R., Thakkar S., Czarnaski J., and Schott B.: "Power-Aware Acoustic Beamforming," International Military Sensing Symposium, 2003.

[34] Scrofano R., Choi S., and Prasanna V. K.: "Energy Efficiency of FPGAs and Programmable Processors for Matrix Multiplication," IEEE International Conference on Field Programmable Technology, 2002.

[35] Scrofano R., Mohanty S., Prasanna V. K., Bogdanowicz J. F., and Wanek E. W.: "Memory Configuration based Design Space Exploration using MILAN," High Performance Embedded Computing, 2004.

[36] SimpleScalar Tool Suite. http://www.simplescalar.com/

[37] SimplePower: http://www.cse.psu.edu/~mdl/software.htm

[38] Singer P.: "The Optimal Detector," SPIE Conference: Signal and Data Processing for Small Targets, 2002.

[39] Sinha A. and Chandrakasan A.: "JouleTrack-A Web Based Tool For Software Energy Profiling," Design Automation Conference, 2001.

[40] Sztipanovits J. and Karsai G.: "Model-Integrated Computing", *Computer*, Apr. 1997, pg. 110-112.

[41] TI C5000 Series DSP. http://dspvillage.ti.com/

[42] Warmer D. G. and Kleppe A. G.: The Object Constraint Language : Precise Modeling With UML, Addison-Wesley, 1999.

[43] Xilinx Virtex-II Pro Series of devices. http://www.xilinx.com/

[44] Xilinx Xpower – Power Estimator for Xilin FPGAs. http://www.xilinx.com/